Improved Set-Based Symbolic Algorithms for Parity Games*†

Krishnendu Chatterjee[1], Wolfgang Dvořák[2], Monika Henzinger[3], and Veronika Loitzenbauer[4]

1 **IST, Klosterneuburg, Austria**
2 **TU Wien, Vienna, Austria; and University of Vienna, Vienna, Austria**
3 **University of Vienna, Vienna, Austria**
4 **Bar-Ilan University, Ramat Gan, Israel; and University of Vienna, Vienna, Austria**

Abstract

Graph games with ω-regular winning conditions provide a mathematical framework to analyze a wide range of problems in the analysis of reactive systems and programs (such as the synthesis of reactive systems, program repair, and the verification of branching time properties). Parity conditions are canonical forms to specify ω-regular winning conditions. Graph games with parity conditions are equivalent to μ-calculus model checking, and thus a very important algorithmic problem. Symbolic algorithms are of great significance because they provide scalable algorithms for the analysis of large finite-state systems, as well as algorithms for the analysis of infinite-state systems with finite quotient. A set-based symbolic algorithm uses the basic set operations and the one-step predecessor operators. We consider graph games with n vertices and parity conditions with c priorities (equivalently, a μ-calculus formula with c alternations of least and greatest fixed points). While many explicit algorithms exist for graph games with parity conditions, for set-based symbolic algorithms there are only two algorithms (notice that we use space to refer to the number of sets stored by a symbolic algorithm):

(a) the basic algorithm that requires $O(n^c)$ symbolic operations and linear space; and (b) an improved algorithm that requires $O(n^{c/2+1})$ symbolic operations but also $O(n^{c/2+1})$ space (i.e., exponential space). In this work we present two set-based symbolic algorithms for parity games:

(b) our first algorithm requires $O(n^{c/2+1})$ symbolic operations and only requires linear space; and (b) developing on our first algorithm, we present an algorithm that requires $O(n^{c/3+1})$ symbolic operations and only linear space.

We also present the first linear space set-based symbolic algorithm for parity games that requires at most a sub-exponential number of symbolic operations.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases model checking, graph games, parity games, symbolic computation, progress measure

Digital Object Identifier 10.4230/LIPIcs.CSL.2017.18

* All authors are partially supported by the Vienna Science and Technology Fund (WWTF) through project ICT15-003. K. C. is partially supported by the Austrian Science Fund (FWF) NFN Grant No S11407-N23 (RiSE/SHiNE) and an ERC Start grant (279307: Graph Games). W. D., M. H., and V. L. received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506. V. L. is partially supported by the ISF grant #1278/16 and an ERC Consolidator Grant (project MPM).

† Some proofs are omitted due to space restrictions, but a full version with all proofs is available at [12].

1 Introduction

In this work we present improved set-based symbolic algorithms for solving graph games with parity winning conditions, which is equivalent to modal μ-calculus model-checking.

Graph games. Two-player graph games provide the mathematical framework to analyze several important problems in computer science, especially in formal methods for the analysis of reactive systems. Graph games are games that proceed for an infinite number of rounds, where the two players take turns to move a token along the edges of the graph to form an infinite sequence of vertices (which is called a *play* or a *trace*). The desired set of plays is described as an ω-regular winning condition. A *strategy* for a player is a recipe that describes how the player chooses to move tokens to extend plays, and a *winning* strategy ensures the desired set of plays against all strategies of the opponent. Some classical examples of graph games in formal methods are as follows:

(a) If the vertices and edges of a graph represent the states and transitions of a reactive system, resp., then the synthesis problem (Church's problem [14]) asks for the construction of a *winning strategy* in a graph game [8, 36, 35, 33, 34].

(b) The problems of
  - **(i)** verification of a branching-time property of a reactive system [19], where one player models the existential quantifiers and the opponent models the universal quantifiers; as well as
  - **(ii)** verification of open systems [2], where one player represents the controller and the opponent represents the environment;

  are naturally modeled as graph games, where the winning strategies represent the choices of the existential player and the controller, respectively.

Moreover, game-theoretic formulations have been used for refinement [25], compatibility checking [17] of reactive systems, program repair [28], and synthesis of programs [11]. Graph games with *parity winning* conditions are particularly important since all ω-regular winning conditions (such as safety, reachability, liveness, fairness) as well as all Linear-time Temporal Logic (LTL) winning conditions can be translated to parity conditions [37, 38], and parity games are equivalent to modal μ-calculus model checking [19]. In a parity winning condition, every vertex is assigned a non-negative integer priority from $\{0, 1, \ldots, c-1\}$, and a play is winning if the highest priority visited infinitely often is even. Graph games with parity conditions can model all the applications mentioned above, and there is a rich literature on the algorithmic study of finite-state parity games [19, 6, 40, 29, 43, 31, 39].

Explicit vs. symbolic algorithms. The algorithms for parity games can be classified broadly as *explicit* algorithms, where the algorithms operate on the explicit representation of the graph game, and *implicit or symbolic* algorithms, where the algorithms only use a set of predefined operations and do not explicitly access the graph game. Symbolic algorithms are of great significance for the following reasons:

(a) first, symbolic algorithms are required for large finite-state systems that can be succinctly represented implicitly (e.g., programs with Boolean variables) and symbolic algorithms are scalable, whereas explicit algorithms do not scale; and

(b) second, for infinite-state systems (e.g., real-time systems modeled as timed automata, or hybrid systems, or programs with integer domains) only symbolic algorithms are applicable, rather than explicit algorithms. Hence for the analysis of large systems or infinite-state systems symbolic algorithms are necessary.

Significance of set-based symbolic algorithms. The most significant class of symbolic algorithms for parity games are based on *set operations*, where the allowed symbolic operations are:

(a) basic set operations such as union, intersection, complement, and inclusion; and
(b) one step predecessor (Pre) operations.

Note that the basic set operations (that only involve state variables) are much cheaper as compared to the predecessor operations (that involve both variables of the current and of the next state). Thus in our analysis we will distinguish between the basic set operations and the predecessor operations. We refer to the number of sets stored by a set-based symbolic algorithm as its space. The significance of set-based symbolic algorithms is as follows:

(a) First, in several domains of the analysis of both infinite-state systems (e.g., games over timed automata or hybrid systems) as well as large finite-state systems (e.g., programs with many Boolean variables, or bounded integer variables), the desired model-checking question is specified as a μ-calculus formula with the above set operations [18, 16]. Thus an algorithm with the above set operations provides a symbolic algorithm that is directly applicable to the formal analysis of such systems.
(b) Second, in other domains such as in program analysis, the one-step predecessor operators are routinely used (namely, with the weakest-precondition as a predicate transformer). A symbolic algorithm based only on the above operations thus can easily be developed on top of the existing implementations. Moreover, recent work [4] shows how efficient procedures (such as constraint-based approaches using SMTs) can be used for the computation of the above operations in infinite-state games. This highlights that symbolic one-step operations can be applied to a large class of problems.
(c) Finally, if a symbolic algorithm is described with the above very basic set of operations, then any practical improvement to these operations in a particular domain would translate to a symbolic algorithm that is faster in practice for the respective domain.

Thus the problem is practically relevant, and understanding the symbolic complexity of parity games is an interesting and important problem.

Previous results. We summarize the main previous results for finite-state game graphs with parity conditions. Consider a parity game with n vertices, m edges, and c priorities (which is equivalent to μ-calculus model-checking of transitions systems with n states, m transitions, and a μ-calculus formula of alternation depth c). In the interest of concise presentation, in the following discussion, we ignore denominators in c in the running time bounds, see Theorems 7 and 8, and the references for precise bounds.

Let us first consider *set-based symbolic algorithms*. Recall that we use space to refer to the number of sets stored by a symbolic algorithm. The basic set-based symbolic algorithm (based on the direct evaluation of the nested fixed point of the μ-calculus formula) for parity games requires $O(n^c)$ symbolic operations and space linear in c [20]. In a breakthrough result [6], a new set-based symbolic algorithm was presented that requires $O(n^{c/2+1})$ symbolic operations, but also requires $O(n^{c/2+1})$ many sets, i.e., exponential space as compared to the linear space of the basic algorithm. A simplification of the result of [6] was presented in [40].

Now consider *explicit algorithms* for parity games. The classical algorithm requires $O(n^{c-1}m)$ time and can be implemented in quasi-linear space [44, 34], which was then improved to the small-progress measure algorithm that requires $O(n^{c/2}m)$ time and space to store $O(c \cdot n)$ integer counters [29]. The small-progress measure algorithm, which is an explicit algorithm, uses an involved domain of the product of integer priorities and *lift* operations (which is a lexicographic max and min in the involved domain). The algorithm shows that the fixed point of the lift operation computes the solution of the parity game.

The lift operation can be encoded with algebraic binary decision diagrams [9] but this does not provide a set-based symbolic algorithm. Other notable explicit algorithms for parity games are as follows:

(a) a strategy improvement algorithm [43], which in the worst-case is exponential [22];

(b) a dominion-based algorithm [31] that requires $n^{O(\sqrt{n})}$ time and a randomized $n^{O(\sqrt{n/\log n})}$ algorithm [5] (both algorithms are sub-exponential, but inherently explicit algorithms); and, combining the small-progress measure and the dominion-based algorithm,

(c) an $O(n^{c/3}m)$ time algorithm [39] and its improvement for dense graphs with c sub-polynomial in n to an $O(n^{c/3}n^{4/3})$ time algorithm [13] (both bounds are simplified).

A recent breakthrough result [10] shows that parity games can be solved in $O(n^{\log c})$ time, i.e., quasi-polynomial time. Follow-up work [30, 21] reduced the space requirements from quasi-polynomial to $O(n \log n \log c)$, i.e., to quasi-linear, space.

While the above algorithms are specified for finite-state graphs, the symbolic algorithms also apply to infinite-state graphs with a finite bi-simulation quotient (such as timed-games, or rectangular hybrid games), and then n represents the size of the finite quotient.

Our contributions. Our results for game graphs with n vertices and parity objectives with c priorities are as follows.

1. First, we present a set-based symbolic algorithm that requires $O(n^{c/2+1})$ symbolic operations and linear space (i.e., a linear number of sets). Thus it matches the symbolic operations bound of [6, 40] and brings the space requirements down to a linear number of sets as in the classical algorithm (albeit linear in n and not in c).
2. Second, developing on our first algorithm, we present a set-based symbolic algorithm that requires $O(n^{c/3+1})$ symbolic operations (simplified bound) and linear space. Thus it improves the symbolic operations of [6, 40] while achieving an exponential improvement in the space requirement. We also present a modification of our algorithm that requires $n^{O(\sqrt{n})}$ symbolic operations and at most linear space. This is the first linear-space set-based symbolic algorithm that requires at most a sub-exponential number of symbolic operations.

In the results above the number of symbolic operations mentioned is the number of predecessor operations, and in all cases the number of required basic set operations (which are usually cheaper) is at most a factor of $O(n)$ more. Our main results and comparison with previous set-based symbolic algorithms are presented in the table below.

reference	symbolic operations	space
[20, 44]	$O(n^c)$	$O(c)$
[6, 40]	$O(n^{c/2+1})$	$O(n^{c/2+1})$
Thm. 7	$\mathbf{O(n^{c/2+1})}$	$\mathbf{O(n)}$
Thm. 8	$\min\{\mathbf{n^{O(\sqrt{n})}}, \mathbf{O(n^{c/3+1})}\}$	$\mathbf{O(n)}$

Our *technical contributions* are as follows. We provide a symbolic version of the progress measure algorithm. The main challenge is to succinctly encode the numerical domain of the progress measure as sets. More precisely, the challenge is to represent $\Theta(n^{c/2})$ many numerical values with $O(n)$ many sets, such that they can still be efficiently processed by a set-based symbolic algorithm. For the sake of efficiency our algorithms consider sets S_r storing all vertices with progress measure at least r. However, there are $\Theta(n^{c/2})$ many such sets S_r and thus, to reduce the space requirements to a linear number of sets, we use a succinct representation that encodes all the sets S_r with just $O(n)$ many sets, such that we can restore a set S_r efficiently whenever it is processed by the algorithm.

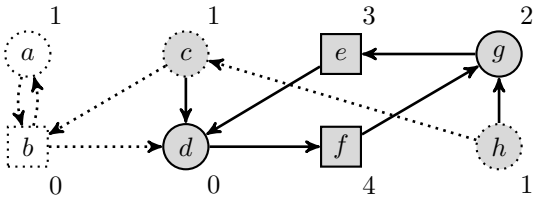

Figure 1 A parity game with 5 priorities. Circles denote player $\mathcal{E}$ vertices, squares denote player $\mathcal{O}$ vertices. The numeric label of a vertex gives its priority, e.g., a is an $\mathcal{E}$-vertex with priority 1. The set of solid vertices, i.e., the set $\{d, e, f, g\}$, is a player-$\mathcal{E}$ dominion and the union of this $\mathcal{E}$-dominion with the vertices c and h is the winning set of player $\mathcal{E}$ in this game. The solid edges indicate a winning strategy for player $\mathcal{E}$.

2 Preliminaries and Previous Results

2.1 Parity Games

We consider *games on graphs* played by two adversarial players, denoted by $\mathcal{E}$ (for even) and $\mathcal{O}$ (for odd). We use z to denote one of the players of $\{\mathcal{E}, \mathcal{O}\}$ and $\overline{z}$ to denote its opponent. We denote by (V, E) a directed graph with $n = |V|$ vertices and $m = |E|$ edges, where V is the vertex set and E is the edge set. A *game graph* $\mathcal{G} = ((V, E), (V_{\mathcal{E}}, V_{\mathcal{O}}))$ is a directed graph (V, E) with a partition of the vertices into player-$\mathcal{E}$ vertices $V_{\mathcal{E}}$ and player-$\mathcal{O}$ vertices $V_{\mathcal{O}}$. For a vertex $u \in V$, we write $\mathit{Out}(u) = \{v \in V \mid (u, v) \in E\}$ for the set of successor vertices of u. As a standard convention (for technical simplicity) we consider that every vertex has at least one outgoing edge, i.e., $\mathit{Out}(u)$ is non-empty for all vertices u.

A game is initialized by placing a token on a vertex. Then the two players form an infinite path, called *play*, in the game graph by moving the token along the edges. Whenever the token is on a vertex of V_z, player z moves the token along one of the outgoing edges of the vertex. Formally, a *play* is an infinite sequence $\langle v_0, v_1, v_2, \ldots \rangle$ of vertices such that $(v_j, v_{j+1}) \in E$ for all $j \geq 0$.

A *parity game* $\mathcal{P} = (\mathcal{G}, \alpha)$ with c priorities consists of a game graph $G = ((V, E), (V_{\mathcal{E}}, V_{\mathcal{O}}))$ and a *priority function* $\alpha : V \to [c]$ that assigns an integer from the set $[c] = \{0, \ldots, c-1\}$ to each vertex (see Figure 1 for an example). Player $\mathcal{E}$ (resp. player $\mathcal{O}$) wins a play of the parity game if the *highest* priority occurring infinitely often in the play is even (resp. odd). We denote by P_i the set of vertices with priority i, i.e., $P_i = \{v \in V \mid \alpha(v) = i\}$. Note that if P_i is empty for $0 < i < c-1$, then the priorities $> i$ can be decreased by 2 without changing the parity condition, and when P_{c-1} is empty, we simply have a parity game with a priority less; thus we assume w.l.o.g. $P_i \neq \emptyset$ for $0 < i < c$.

A *strategy* of a player $z \in \{\mathcal{E}, \mathcal{O}\}$ is a function that, given a finite prefix of a play ending at $v \in V_z$, selects a vertex from $\mathit{Out}(v)$ to extend the finite prefix. *Memoryless strategies* depend only on the last vertex of the finite prefix. That is, a memoryless strategy of player z is a function $\sigma : V_z \to V$ such that for all $v \in V_z$ we have $\sigma(v) \in \mathit{Out}(v)$. It is well-known that for parity games it is sufficient to consider memoryless strategies [19, 34]. Therefore we only consider memoryless strategies from now on. A start vertex v, a strategy σ for $\mathcal{E}$, and a strategy π for $\mathcal{O}$ describe a unique play $\omega(v, \sigma, \pi) = \langle v_0, v_1, v_2, \ldots \rangle$, which is defined as follows: $v_0 = v$ and for all $i \geq 0$, if $v_i \in V_{\mathcal{E}}$, then $\sigma(v_i) = v_{i+1}$, and if $v_i \in V_{\mathcal{O}}$, then $\pi(v_i) = v_{i+1}$.

A strategy σ is *winning* for player $\mathcal{E}$ at start vertex v iff for all strategies π of player $\mathcal{O}$ we have that the play $\omega(v, \sigma, \pi)$ satisfies the parity condition, and analogously for winning strategies for player $\mathcal{O}$. A vertex v belongs to the *winning set* W_z of player z if player z has

a winning strategy from start vertex v. Every vertex is winning for exactly one of the two players. The algorithmic problem we study for parity games is to compute the winning sets of the two players. A non-empty set of vertices D is a *player-z dominion* if player z has a winning strategy from every vertex of D that also ensures only vertices of D are visited.

2.2 Set-based Symbolic Operations

Symbolic algorithms operate on sets of vertices, which are usually described by Binary Decision Diagrams (BDD) [32, 1]. For the symbolic algorithms for parity games we consider the most basic form of symbolic operations, namely, *set-based symbolic* operations. More precisely, we only allow the following operations:

Basic set operations. First, we allow basic set operations like $\cup$, $\cap$, $\setminus$, $\subseteq$, and $=$.

One-step operations. Second, we allow the following symbolic one-step operations:

(a) the one-step predecessor operator $\mathsf{Pre}(B) = \{v \in V \mid \exists u \in B : (v,u) \in E\}$; and

(b) the one-step *controllable* predecessor operator $\mathsf{CPre}_z(B) = \{v \in V_z \mid \mathit{Out}(v) \cap B \neq \emptyset\} \cup \{v \in V_{\overline{z}} \mid \mathit{Out}(v) \subseteq B\}$; i.e., the CPre_z operator computes all vertices from which z can ensure that in the next step the successor belongs to the given set B. Moreover, the CPre_z operator can be defined using the Pre operator and basic set operations as follows: $\mathsf{CPre}_z(B) = \mathsf{Pre}(B) \setminus (V_{\overline{z}} \cap \mathsf{Pre}(V \setminus B))$.

Algorithms that use only the above operations are called *set-based symbolic* algorithms. Additionally, successor operations can be allowed but are not needed for our algorithms. The above symbolic operations correspond to primitive operations in standard symbolic packages like CuDD [41].

Typically, the basic set operations are cheaper (as they encode relationships between state variables) as compared to the one-step symbolic operations (which encode the transitions and thus the relationship between the variables of the present and of the next state). Thus in our analysis we distinguish between these two types of operations.

For the *space* requirements of set-based symbolic algorithms, as per standard convention [6, 9], we consider that a set is stored in constant space (e.g., a set can be represented symbolically as one BDD [7]). We thus consider the space requirement of a symbolic algorithm to be the maximal number of sets that the algorithm has to store.

2.3 Progress Measure Algorithm

We first provide basic intuition for the progress measure [29] and then provide the formal definitions. Solving parity games can be reduced to computing the progress measure [29]. In Section 3 we present a set-based symbolic algorithm to compute the progress measure.

High-level intuition. Towards a high-level intuition behind the progress measure, consider an $\mathcal{E}$-dominion D, i.e., player $\mathcal{E}$ wins on all vertices of D without leaving D. Fix a play started at a vertex $u \in D$ in which player $\mathcal{E}$ follows her winning strategy on D. In the play from some point on the highest priority visited by the play, say α^*, has to be even. Let v_* be the vertex after which the highest visited priority is α^* (recall that memoryless strategies are sufficient for parity games). Before v_* is visited, the play might have visited vertices with odd priority higher than α^* but the number of these vertices has to be less than n. The *progress measure* is based on a so-called *lexicographic ranking* function that assigns a *rank* to each vertex v, where the rank is a "vector of counters" for the number of times player $\mathcal{O}$ can force a play to visit an odd priority vertex before a vertex with higher even priority is reached. If player $\mathcal{O}$ can ensure a counter value of at least n, then she can ensure that a

cycle with highest priority odd is reached from v and therefore player $\mathcal{E}$ cannot win from the vertex v. Conversely, if player $\mathcal{O}$ can reach a cycle with highest priority odd before reaching a higher even priority, then she can also force a play to visit an odd priority n times (thus a counter value of n) before reaching a higher even priority. In other words, a vertex u is in the $\mathcal{E}$-dominion D if and only if player $\mathcal{O}$ *cannot* force any counter value to reach n from u. When a vertex u is classified as winning for player $\mathcal{O}$, it is marked with the rank $\top$ and whenever $\mathcal{O}$ has a strategy for some vertex v to reach a $\top$-ranked vertex, it is also winning for player $\mathcal{O}$ and thus ranked $\top$. Computing the progress measure is done by updating the rank of a vertex according to the ranks of its successors and is equal to computing the least simultaneous fixed point for all vertices with respect to "ranking functions".

An additional property of the progress measure is that the ranks assigned to the vertices of the $\mathcal{E}$-dominion provide a certificate for a winning strategy of player $\mathcal{E}$ within the dominion, namely, player $\mathcal{E}$ can follow edges that lead to vertices with "lower or equal" rank with respect to a specific ordering of the ranks.

Formal definitions. We next provide formal definitions of *rank*, the *ranking function*, the ordering on the ranks, the *lift*-operators, and finally the *progress measure* (see also [29]).

We start with *the progress measure domain* M_G^∞ and consider parity games with n vertices and priorities $[c]$. Let n_i be the number of vertices with priority i for odd i (i.e., $n_i = |P_i|$), let $n_i = 0$ for even i, and let $N_i = [n_i + 1]$ for $0 \leq i < c$. Let $M_{\mathcal{G}} = (N_0 \times N_1 \times \cdots \times N_{c-2} \times N_{c-1})$ be the product domain where every even index is 0 and every odd index i is a number between 0 and n_i. The *progress measure domain* is $M_{\mathcal{G}}^\infty = M_{\mathcal{G}} \cup \{\top\}$, where $\top$ is a special element called the top element. Then we have $|M_{\mathcal{G}}^\infty| = 1 + \prod_{i=1}^{\lfloor c/2 \rfloor}(n_{2i-1}+1) = O\big(\big(\frac{n}{\lfloor c/2 \rfloor}\big)^{\lfloor c/2 \rfloor}\big)$ [29] (this bound uses that w.l.o.g. $|P_i| > 0$ for each priority $i > 0$).

A *ranking function* $\rho : V \to M_{\mathcal{G}}^\infty$ assigns to each vertex a *rank* r that is either one of the c dimensional vectors in $M_{\mathcal{G}}$ or the top element $\top$. Note that a rank has at most $\lfloor c/2 \rfloor$ non-zero entries. Informally, we call the entries of a rank with an odd index i a "counter" because as long as the top element is not reached, it counts (with "carry", i.e., if n_i is reached, the next highest counter is increased by one and the counter at index i is reset to zero) the number of times a vertex of priority i is reached before a vertex of higher priority is reached (from some specific start vertex). The co-domain of ρ is $M_{\mathcal{G}}^\infty = M_{\mathcal{G}} \cup \{\top\}$ and we index the elements of the vectors from 0 to $c-1$.

We use the *lexicographic comparison operator* $<$ of the ranks assigned by ρ: the vectors are considered in the lexicographical order, where the left most entry is the least significant one and the right most entry is the most significant one, and $\top$ is the maximum element of the ordering. We write $\bar{0}$ to refer to the all zero vector (i.e., the minimal element of the ordering) and $\bar{N}$ to refer to the maximal vector $(n_0, n_1, \ldots, n_{c-1})$ (i.e., the second largest element, after $\top$, in the ordering).

Next we introduce the *lexicographic increment and decrement operations.* Given a rank r, i.e., either a vector or $\top$, we refer to the successor in the ordering $<$ by $\mathrm{inc}(r)$ (with $\mathrm{inc}(\top) = \top$), and to the predecessor in the ordering $<$ by $\mathrm{dec}(r)$ (with $\mathrm{dec}(\bar{0}) = \bar{0}$). We also consider restrictions of inc and dec to fewer dimensions, which are described below. Given a vector $x = (x_0, x_1, x_2, \ldots, x_{c-1})$, we denote by $\langle x \rangle_\ell$ (for $0 \leq \ell < c$) the vector $(0, 0, \ldots, 0, x_\ell, \ldots, x_{c-1})$, where we set all elements with index less than ℓ to 0; in particular $x = \langle x \rangle_0$. Intuitively, we use the notation $\langle x \rangle_\ell$ to "reset the counters" for priorities lower than ℓ when a vertex of priority ℓ is reached (as long as we have not counted up to the top element). Moreover, we also generalize the ordering to a family of orderings $<_\ell$ where $x <_\ell y$ for two vectors x and y iff $\langle x \rangle_\ell < \langle y \rangle_\ell$; the top element $\top$ is the maximum element of each

ordering. In particular, $x <_0 y$ iff $x < y$ and in our setting also $x <_1 y$ iff $x < y$. We further have restricted versions inc_ℓ and dec_ℓ of inc and dec; note that dec_ℓ is a partial function and that ℓ will be the priority of the vertex v for which we want to update its rank and x will be the rank of one of its neighbors in the game graph.

- $\mathrm{inc}_\ell(x)$: For $x = \top$ we have $\mathrm{inc}_\ell(\top) = \top$; Otherwise $\mathrm{inc}_\ell(x) = \langle x \rangle_\ell$ if ℓ is even and $\mathrm{inc}_\ell(x) = \min\{y \in M_\mathcal{G}^\infty \mid y >_\ell x\}$ if ℓ is odd.
- $\mathrm{dec}_\ell(x) : \mathrm{dec}_\ell(x) = \bar{0}$ if $\langle x \rangle_\ell = \bar{0}$; Otherwise if $\langle x \rangle_\ell > \bar{0}$ then $\mathrm{dec}_\ell(x) = \min\{y \in M_\mathcal{G} \mid x = \mathrm{inc}_\ell(y)\}$.

For $\bar{0} < \langle x \rangle_\ell < \top$ we have $\mathrm{inc}_\ell(\mathrm{dec}_\ell(x)) = \mathrm{dec}_\ell(\mathrm{inc}_\ell(x)) = \langle x \rangle_\ell$ while for $\top$ we only have $\mathrm{inc}_\ell(\mathrm{dec}_\ell(\top)) = \top$ and for $\langle x \rangle_\ell = \bar{0}$ only $\mathrm{dec}_\ell(\mathrm{inc}_\ell(x)) = \bar{0}$. By the restriction of inc by the priority ℓ of v, for both even and odd priorities the counters for lower (odd) priorities are reset to zero as long as the top element is not reached. For an odd ℓ additionally the counter for ℓ is increased or, if the counter for ℓ has already been at n_ℓ, then one of the higher counters is increased while the counter for ℓ is reset to zero as well; if no higher counter can be increased any more, then the rank of v is set to $\top$.

Recall the interpretation of the progress measure as a witness for a player-$\mathcal{E}$ winning strategy on an $\mathcal{E}$-dominion, where player $\mathcal{E}$ wants to follow a path of non-increasing rank. The function best we define next reflects the ability of player $\mathcal{E}$ to choose the edge leading to the lowest rank when he owns the vertex, while for player-$\mathcal{O}$ vertices all edges need to lead to non-increasing ranks if player $\mathcal{E}$ can win from this vertex. The function best for each vertex v and ranking function ρ is given by

$$\mathrm{best}(\rho, v) = \begin{cases} \min\{\rho(w) \mid (v,w) \in E\} & \text{if } v \in V_\mathcal{E}\,, \\ \max\{\rho(w) \mid (v,w) \in E\} & \text{if } v \in V_\mathcal{O}\,. \end{cases}$$

Finally, the lift operation implements the incrementing of the rank of a vertex v according to its priority and the ranks of its neighbors:

$$\mathrm{Lift}(\rho, v)(u) = \begin{cases} \mathrm{inc}_{\alpha(v)}(\mathrm{best}(\rho, v)) & \text{if } u = v\,, \\ \rho(u) & \text{otherwise}\,. \end{cases}$$

The $\mathrm{Lift}(., v)$-operators are monotone and the *progress measure* for a parity game is defined as the *least simultaneous fixed point of all* $\mathrm{Lift}(., v)$*-operators.* The progress measure can be computed by starting with the ranking function equal to the all-zero function and iteratively applying the $\mathrm{Lift}(., v)$-operators in an arbitrary order [29]. Note that in this case the $\mathrm{Lift}(., v)$-operator assigns only rank vectors r with $r = \langle r \rangle_{\alpha(v)}$ to v. See [29] for a worst-case example for any lifting algorithm. By [29], the winning set of player $\mathcal{E}$ can be obtained from the progress measure by selecting those vertices whose rank is a vector, i.e., smaller than $\top$.

▶ **Lemma 1** ([29]). *For a given parity game and the progress measure ρ with co-domain $M_\mathcal{G}^\infty$, the set of vertices with $\rho(v) < \top$ is exactly the winning set of player $\mathcal{E}$.*

This implies that to solve parity games it is sufficient to provide an algorithm that computes the least simultaneous fixed point of all $\mathrm{Lift}(., v)$-operators. The Lift operation can be computed explicitly in $O(m)$ time, which gives the SmallProgressMeasure algorithm of [29]. The SmallProgressMeasure algorithm is an explicit algorithm that requires $O(m \cdot |M_\mathcal{G}^\infty|) = O\big(m \cdot \big(\frac{n}{\lfloor c/2 \rfloor}\big)^{\lfloor c/2 \rfloor}\big)$ time and $O(n \cdot c)$ space (assuming constant size integers).

3 Set-based Symbolic Progress Measure Algorithm for Parity Games

In this section we present a set-based symbolic algorithm for parity games, with n vertices and c priorities, by showing how to compute a progress measure (see Section 2.3) using only set-based symbolic operations (see Section 2.2). All proofs are provided in Appendix A. We mention the *key differences* of Algorithm SymbolicParityDominion and the explicit progress-measure algorithm ([29], see Section 2.3).

1. The main challenge for an efficient set-based symbolic algorithm similar to the SmallProgressMeasure algorithm is to represent $\Theta(n^{c/2})$ many numerical values succinctly with $O(n)$ many sets, such that they can still be efficiently processed by a symbolic algorithm.
2. To exploit the power of symbolic operations, in each iteration of the algorithm we compute all vertices whose rank can be increased to a certain value r. This is in sharp contrast to the explicit progress-measure algorithm, where vertices are considered one by one and the rank is increased to the maximal possible value.

Key concepts. Recall that the progress measure for parity games is defined as the least simultaneous fixed point of the Lift(ρ, v)-operators on a ranking function $\rho : V \to M_{\mathcal{G}}^{\infty}$. There are two key aspects of our algorithm:

1. *Symbolic encoding of numerical domain.* In our symbolic algorithm we cannot directly deal with the ranking function but have to use sets of vertices to encode it. We first formulate our algorithm with sets S_r for $r \in M_{\mathcal{G}}^{\infty}$ that contain all vertices that have rank r or higher; that is, given a function ρ, the corresponding sets are $S_r = \{v \mid \rho(v) \geq r\}$. On the other hand, given a family of sets $\{S_r\}_r$, the corresponding ranking function $\rho_{\{S_r\}_r}$ is given by $\rho_{\{S_r\}_r}(v) = \max\{r \in M_{\mathcal{G}}^{\infty} \mid v \in S_r\}$. This formulation encodes the numerical domain with sets but uses exponential in c many sets.
2. *Space efficiency.* We refine the algorithm to directly encode the ranks with one set for each possible index-value pair. This reduces the required number of sets to linear at the cost of increasing the number of set operations only by a factor of n; the number of one-step symbolic operations does not increase.

We first present the variant that uses an exponential number of sets and then show how to reduce the number of sets to linear.

The above ideas yield a set-based symbolic algorithm, but since we now deal with sets of vertices, as compared to individual vertices, the correctness needs to be established. The non-trivial aspect of the proof is to identify appropriate *invariants on sets* (which we call *symbolic invariants*, see Invariant 3) and use them to establish the correctness.

3.1 The Set-based Symbolic Progress Measure Algorithm

The codomain M_h^{∞}. We formulate our algorithm such that it cannot only compute the winning sets of the players but also $\mathcal{E}$-dominions of size at most $h+1$. (For $\mathcal{O}$-dominions add one to each priority and exchange the roles of the two players.) The only change needed for this is to use the codomain M_h^{∞}, instead of $M_{\mathcal{G}}^{\infty}$, for the inc and dec operations. The codomain M_h^{∞} contains all ranks of $M_{\mathcal{G}}^{\infty}$ whose entries sum up to at most h.

The sets S_r and the ranking function $\rho_{\{S_r\}_r}$. The algorithm implicitly maintains a rank for each vertex. A vertex is contained in a set S_r only if its maintained rank is *at least* r. Each set S_r is monotonically increasing throughout the algorithm. The rank of a vertex v is the highest r such that $v \in S_r$. In other words, the family of sets $\{S_r\}_r$ defines the ranking function $\rho_{\{S_r\}_r}(v) = \max\{r \in M_h^{\infty} \mid v \in S_r\}$. When the rank of a vertex is increased,

Algorithm SymbolicParityDominion: Symbolic Progress Measure Algorithm

Input : *parity game* $\mathcal{P} = (\mathcal{G}, \alpha)$, with *game graph* $\mathcal{G} = ((V, E), (V_\mathcal{E}, V_\mathcal{O}))$, *priority function* $\alpha : V \to [c]$, and *parameter* $h \in [0, n] \cap \mathbb{N}$
Output : Set containing all $\mathcal{E}$-dominions of size $\leq h + 1$, which is an $\mathcal{E}$-dominion or empty.

```
1  S_0̄ ← V; S_r ← ∅ for r ∈ M_h^∞ \ {0̄};
2  r ← inc(0̄);
3  while true do
4      if r ≠ ⊤ then
5          Let ℓ be maximal such that r = ⟨r⟩_ℓ;
6          S_r ← S_r ∪ ⋃_{1≤k≤(ℓ+1)/2} (CPre_O(S_{dec_{2k−1}(r)}) ∩ P_{2k−1});
7          repeat
8              S_r ← S_r ∪ (CPre_O(S_r) \ ⋃_{ℓ<k<c} P_k)
9          until a fixed-point for S_r is reached;
10     else if r = ⊤ then
11         S_⊤ ← S_⊤ ∪ ⋃_{1≤k≤⌊c/2⌋} (CPre_O(S_{dec_{2k−1}(⊤)}) ∩ P_{2k−1});
12         repeat
13             S_⊤ ← S_⊤ ∪ (CPre_O(S_⊤))
14         until a fixed-point for S_⊤ is reached;
15     r′ ← dec(r);
16     if S_{r′} ⊇ S_r and r < ⊤ then
17         r ← inc(r)
18     else if S_{r′} ⊇ S_r and r = ⊤ then
19         break
20     else
21         repeat
22             S_{r′} ← S_{r′} ∪ S_r;
23             r′ ← dec(r′);
24         until S_{r′} ⊇ S_r;
25         r ← inc(r′);
26 return V \ S_⊤
```

this information has to be propagated to its predecessors. This is achieved efficiently by maintaining *anti-monotonicity* among the sets, i.e., we have $S_{r'} \supseteq S_r$ for all r and all $r' < r$ before and after each iteration. Anti-monotonicity together with defining the sets $S_{r'}$ to contain vertices with rank *at least* r' instead of *exactly* r' enables us to decide whether the rank of a vertex v can be increased to r by only considering one set $S_{r'}$.

Structure of the algorithm. The set $S_{\bar{0}}$ is initialized with the set of all vertices V, while all other sets S_r for $r > \bar{0}$ are initially empty, i.e., the ranks of all vertices are initialized with the zero vector. The variable r is initially set to the second lowest rank $\mathrm{inc}(\bar{0})$ that is one at index 1 and zero otherwise. In the while-loop the set S_r is updated for the value of r at the beginning of the iteration (see below). After the update of S_r, it is checked whether the set corresponding to the next lowest rank already contains the vertices newly added to S_r, i.e., whether the anti-monotonicity is preserved. If the anti-monotonicity is preserved despite the update of S_r, then for $r < \top$ the value of r is increased to the next highest rank and for $r = \top$ the algorithm terminates. Otherwise the vertices newly added to S_r are also added to all sets with $r' < r$ that do not already contain them; the variable r is then updated to the lowest r' for which a new vertex is added to $S_{r'}$ in this iteration.

Update of set S_r. To reach a simultaneous fixed point of the lift-operators, the rank of a vertex v has to be increased to $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v)$ whenever the value of $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v)$ is strictly higher than $\rho_{\{S_r\}_r}(v)$ for the current ranking function $\rho_{\{S_r\}_r}$. Now consider a fixed

iteration of the while-loop and let r be as at the beginning of the while-loop. Let $\rho_{\{S_r\}_r}$ be denoted by ρ for short. In this update of the set S_r we want to add to S_r all vertices v with $\rho(v) < r$ and $\mathrm{Lift}(\rho, v)(v) \geq r$ under the condition that the priority of v allows v to be assigned the rank r, i.e., $r = \langle r \rangle_{\alpha(v)}$. Note that by the anti-monotonicity property the set S_r already contains all vertices with $\rho(v) \geq r$.

1. We first consider the case $r < \top$. Let ℓ be maximal such that $r = \langle r \rangle_\ell$, i.e., the first ℓ entries with indices 0 to $\ell - 1$ of r are 0 and the entry with index ℓ is larger than 0. Note that ℓ is odd. We have that only the $\mathrm{Lift}(., v)$-operators with $\alpha(v) \leq \ell$ can increase the rank of a vertex to r as all the others would set the element with index ℓ to 0.
   Recall that $\mathrm{Lift}(\rho, v)(v) = \mathrm{inc}_{\alpha(v)}(\mathrm{best}(\rho, v))$. The function best is implemented by the $\mathsf{CPre}_{\mathcal{O}}$ operator: For a player-$\mathcal{E}$ vertex the value of best increases only if the ranks of all successor have increased, for a player-$\mathcal{O}$ vertex it increases as soon as the maximum rank among the successor vertices has increased. The function $\mathrm{inc}_{\alpha(v)}(x)$ for $x < \top$ behaves differently for odd and even $\alpha(v)$ (see Section 2.3): If $\alpha(v)$ is odd, then $\mathrm{inc}_{\alpha(v)}(x)$ is the smallest rank y in M_h^∞ such that $y >_{\alpha(v)} x$, i.e., y is larger than x w.r.t. indices $\geq \alpha(v)$. If $\alpha(v)$ is even, then $\mathrm{inc}_{\alpha(v)}(x)$ is equal to x with the indices lower than $\alpha(v)$ set to 0.
   - **(i)** First, consider a $\mathrm{Lift}(\rho, v)$ operation with odd $\alpha(v) \leq \ell$, i.e., let $\alpha(v) = 2k - 1$ for some $1 \leq k \leq (\ell+1)/2$. Then $\mathrm{Lift}(\rho, v)(v) \geq r$ only if (a) $v \in V_{\mathcal{E}}$ and all successors w have $\rho(w) \geq \mathrm{dec}_{2k-1}(r)$, or (b) $v \in V_{\mathcal{O}}$ and one successor w has $\rho(w) \geq \mathrm{dec}_{2k-1}(r)$. That is, $\mathrm{Lift}(\rho, v)(v) \geq r$ only if $v \in \mathsf{CPre}_{\mathcal{O}}(S_{\mathrm{dec}_{2k-1}(r)})$. Vice versa, we have that if $v \in \mathsf{CPre}_{\mathcal{O}}(S_{\mathrm{dec}_{2k-1}(r)})$ then by $\rho = \rho_{\{S_r\}_r}$ also $\mathrm{Lift}(\rho, v)(v) \geq r$. This observation is implemented in SymbolicParityDominion in line 6, where such vertices v are added to S_r.
   - **(ii)** Now, consider a $\mathrm{Lift}(\rho, v)$ operation with even $\alpha(v) \leq \ell$, i.e., let $\alpha(v) = 2k$ for some $1 \leq k \leq \ell/2$. Then $\mathrm{Lift}(\rho, v)(v) \geq r$ only if
     - **(a)** $v \in V_{\mathcal{E}}$ and all successors w have $\rho(w) \geq r$, or
     - **(b)** $v \in V_{\mathcal{O}}$ and one successor w has $\rho(w) \geq r$.

     That is, $\mathrm{Lift}(\rho, v)(v) \geq r$ only if $v \in \mathsf{CPre}_{\mathcal{O}}(S_r)$. Vice versa, we have that if $v \in \mathsf{CPre}_{\mathcal{O}}(S_r)$ then $\mathrm{Lift}(\rho, v)(v) \geq r$. In SymbolicParityDominion these vertices are added iteratively in line 8 until a fixed point is reached. The algorithm also adds vertices v with odd priority to S_r, but due do the above argument we have $\mathrm{Lift}(\rho, v)(v) > r$ and thus they can be included in S_r.
2. The case $r = \top$ works similarly except that (a) every vertex is a possible candidate for being assigned the rank $\top$, independent of its priority (line 11), and (b) whenever x is equal to $\top$, $\mathrm{inc}_{\alpha(v)}(x)$ assigns the rank $\top$ independently of $\alpha(v)$ (line 13).

▶ **Example 2.** In this example we apply Algorithm SymbolicParityDominion to the parity game in Figure 1. We have $n_1 = 3$ and $n_3 = 1$ and thus we have to consider ranks in the co-domain $M_G^\infty = \{(0,0), (1,0), (2,0), (3,0), (0,1), (1,1), (2,1), (3,1), \top\}$ (we ignore entires of ranks that are always zero in this notation).

The algorithm initializes the set $S_{(0,0)}$ to $\{a, b, c, d, e, f, g, h\}$ and r to $(1, 0)$. All the other sets S_r are initialized as the empty set. It then proceeds as follows:

1. In the first iteration of the while-loop it processes $r = (1, 0)$. We have $\ell = 1$ and thus the only possible value of k in line 6 is $k = 1$. That is, line 6 adds the vertices in $\mathsf{CPre}_{\mathcal{O}}(S_{0,0}) \cap P_1 = \{a, c, h\}$ to $S_{(1,0)}$ and then in line 8 also b is added. We obtain $S_{(1,0)} = \{a, b, c, h\}$ and as $S_{(1,0)} \subseteq S_{(0,0)}$, the rank r is increased to $(2, 0)$.
2. In the second iteration it processes $r = (2, 0)$ and the vertex a is added to $S_{(2,0)}$ in line 6 and the vertex b is added to $S_{(2,0)}$ in line 8, i.e., $S_{(2,0)} = \{a, b\}$, and r is set to $(3, 0)$.

3. When processing $r = (3,0)$ the set $S_{(3,0)}$ is updated to $\{a, b\}$ and r is increased to $(0,1)$.
4. Now the algorithm processes the rank $(0,1)$ the first time. We have $\ell = 3$ and thus the possible values for k are 1 and 2. The vertex a is added to $S_{(0,1)}$ because it is contained in $\mathsf{CPre}_{\mathcal{O}}(S_{3,0}) \cap P_1$ and the vertex e is added because it is contained in $\mathsf{CPre}_{\mathcal{O}}(S_{0,0}) \cap P_3$ in line 6. Finally, also b and g are added in line 8. That is, we have $S_{(0,1)} = \{a, b, e, g\}$. Now as $S_{(3,0)} \not\subseteq S_{(0,1)}$, $S_{(2,0)} \not\subseteq S_{(0,1)}$ and $S_{(1,0)} \not\subseteq S_{(0,1)}$, we have to decrease r to $(1,0)$, and also to modify the other sets with smaller rank as follows: $S_{(1,0)} = \{a, b, c, e, g, h\}$; and $S_{(2,0)} = S_{(3,0)} = \{a, b, e, g\}$.
5. The algorithm considers $r = (1,0)$ again, makes no changes to $S_{(1,0)}$ and sets r to $(2,0)$.
6. Now considering $r = (2,0)$, the vertex h is added to the set $S_{(2,0)}$ in line 6, i.e., $S_{(2,0)} = \{a, b, e, g, h\}$, and, as h is already contained in $S_{(1,0)}$, r is increased to $(3,0)$.
7. The set $S_{(3,0)}$ is not changed and r is increased to $(0,1)$.
8. The set $S_{(0,1)}$ is not changed and r is increased to $(1,1)$.
9. The vertex a is added to $S_{(1,1)}$ in line 6 and the vertex b is added to $S_{(1,1)}$ in line 8, i.e., $S_{(1,1)} = \{a, b\}$, and r is increased to $(2,1)$.
10. The vertices a, b are added to $S_{(2,1)}$, i.e., $S_{(2,1)} = \{a, b\}$, and r is increased to $(3,1)$.
11. The vertices a, b are added to $S_{(3,1)}$, i.e., $S_{(3,1)} = \{a, b\}$, and r is increased to $\top$.
12. The vertex a is added to $S_\top$ in line 11 and b is added to $S_\top$ in line 13, i.e., $S_\top = \{a, b\}$. Now as $S_{(3,1)} \subseteq S_\top$, the algorithm terminates.

Finally we have that $S_{(0,0)} = \{a, b, c, d, e, f, g, h\}$, $S_{(1,0)} = \{a, b, c, e, g, h\}$, $S_{(2,0)} = \{a, b, e, g, h\}$, $S_{(3,0)} = S_{(0,1)} = \{a, b, e, g\}$, and $S_{(1,1)} = S_{(2,1)} = S_{(3,1)} = S_\top = \{a, b\}$, That is, the algorithm returns $\{c, d, e, f, g, h\}$ as the winning set of player $\mathcal{E}$. The final sets of the algorithm correspond to the progress measure ρ with $\rho(f) = \rho(d) = (0,0)$, $\rho(c) = (1,0)$, $\rho(h) = (2,0)$, $\rho(e) = \rho(g) = (0,1)$, and $\rho(a) = \rho(b) = \top$.

Sketch of bound on number of symbolic operations. Observe that each rank r is considered in at least one iteration of the while-loop but is only reconsidered in a later iteration if at least one vertex was added to the set S_r since the last time r was considered; in this case $O(c)$ one-step operations are performed. Thus the number of symbolic operations per set S_r is of the same order as the number of times a vertex is added to the set. Hence the algorithm can be implemented with $O(c \cdot n \cdot |M_h^\infty|)$ symbolic operations. For the co-domain $M_{\mathcal{G}}^\infty$ the bound $O(c \cdot n \cdot |M_{\mathcal{G}}^\infty|)$ is analogous.

Outline correctness proof. In the following proof we show that when Algorithm SymbolicParityDominion terminates, the ranking function $\rho_{\{S_r\}_r}$ is equal to the progress measure for the given parity game and the co-domain M_h^∞. The same proof applies to the co-domain $M_{\mathcal{G}}^\infty$. The algorithm returns the set of vertices that are assigned a rank $< \top$ when the algorithm terminates. By [39] this set is an $\mathcal{E}$-dominion that contains all $\mathcal{E}$-dominions of size at most $h + 1$ when the co-domain M_h^∞ is used, and by Lemma 1 this set is equal to the winning set of player $\mathcal{E}$ when the co-domain $M_{\mathcal{G}}^\infty$ is used. Thus it remains to show that $\rho_{\{S_r\}_r}$ equals the progress measure for the given co-domain when the algorithm terminates. We show that maintaining the following invariants over all iteration of the algorithm is sufficient for this and then prove that the invariants are maintained. All proofs are in Appendix A and are described for the co-domain M_h^∞.

▶ **Invariant 3** (Symbolic invariants). *In Algorithm SymbolicParityDominion the following three invariants hold. Every rank is from the co-domain M_h^∞ and the* $\mathrm{Lift}(., v)$*-operators are defined w.r.t. the co-domain. Let $\tilde{\rho}$ be the progress measure of the given parity game and let*

$\rho_{\{S_r\}_r}(v) = \max\{r \in M_h^\infty \mid v \in S_r\}$ *be the ranking function with respect to the sets* S_r *that are maintained by the algorithm.*

1. *Before and after each iteration of the while-loop we have that if a vertex* v *is in a set* S_{r_1} *then it is also in* S_{r_2} *for all* $r_2 < r_1$ *(anti-monotonicity).*
2. *Throughout Algorithm SymbolicParityDominion we have* $\tilde{\rho}(v) \geq \rho_{\{S_r\}_r}(v)$ *for all* $v \in V$.
3. *Before and after each iteration of the while-loop we have for the rank stored in* r *and all vertices* v *either* $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) \geq r$ *or* $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) = \rho_{\{S_r\}_r}(v)$. *(b) After the update of* S_r *and before the update of* r *we additionally have* $v \in S_r$ *for all vertices* v *with* $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) = r$ *(closure property).*

The intution behind the invariants is as follows. Invariant 3(1) ensures that the definition of the sets S_r and the ranking function $\rho_{\{S_r\}_r}$ is sound; Invariant 3(2) guarantees that $\rho_{\{S_r\}_r}$ is a lower bound on $\tilde{\rho}$ throughout the algorithm; and Invariant 3(3) shows that when the algorithm terminates, a fixed point of the ranking function $\rho_{\{S_r\}_r}$ with respect to the $\mathrm{Lift}(., v)$-operators is reached. Together these three properties guarantee that when the algorithm terminates the function $\rho_{\{S_r\}_r}$ corresponds to the progress measure, i.e., to the least simultaneous fixed point of the $\mathrm{Lift}(., v)$-operators. We prove the invariants by induction over the iterations of the while-loop. In particular, Invariant 3(1) is ensured by adding vertices newly added to a set S_r also to sets $S_{r'}$ with $r' < r$ that do not already contain them at the end of each iteration of the while-loop. For Invariant 3(2) we show that whenever $\rho_{\{S_r\}_r}(v)$ is increased, i.e., v is added to the set S_r, then no fixed point of the lift-operator for v was reached yet and thus also the progress measure for v has to be at least as high as the new value of $\rho_{\{S_r\}_r}(v)$. The intuition for the proof of Invariant 3(3) is as follows: We first show that $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) = \rho_{\{S_r\}_r}(v)$ remains to hold for all vertices v for which the value of $\rho_{\{S_r\}_r}(v)$ is less than the smallest value r' for which $S_{r'}$ was updated in the considered iteration. In iterations in which the value of the variable r is not increased, this is already sufficient to show part (a) of the invariant. If r is increased, we additionally use part (b) to show part (a). For part (b) we prove by case analysis that, before the update of the variable r, a vertex with $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) = r$ is included in S_r. The correctness of the algorithm then follows from the invariants as outlined above.

3.2 Reducing Space to Linear

Algorithm SymbolicParityDominion requires $|M_{\mathcal{G}}^\infty|$ many sets S_r, which is drastically beyond the space requirement of the progress measure algorithm for explicitly represented graphs. Thus we aim to reduce the space requirement to $O(n)$ many sets in a way that still allows to restore the sets S_r efficiently. For the sake of readability, we assume for this part that c is even. The main idea to reduce the space requirement is as follows.

1. Instead of storing sets S_r corresponding to a specific rank, we encode the value of each coordinate of the rank r separately. That is, we define the sets $C_0^i, C_1^i \ldots, C_{n_i}^i$ for each odd priority i. Intuitively, a vertex is in the set C_x^i iff the i-th coordinate of the rank of v is x. Given these $O(c + n) \in O(n)$ sets, we have encoded the exact rank vector r of each vertex with $r < \top$. To also cover vertices with rank $\top$, we additionally store the set $S_\top$.
2. Whenever the algorithm needs to process a set S_r, we reconstruct it from the stored sets, using a linear number of set operations. Algorithm SymbolicParityDominion has to be adapted as follows. First, at the beginning of each iteration we have to compute the set S_r and up to $c/2$ sets $S_{r'}$ that correspond to some predecessor r' of r. Second, at the end of each iteration we have to update the sets C_x^i to incorporate the updated set S_r.

Computing a set S_r from the sets C_x^i. Let r_i denote the i-th entry of r. To obtain the set $S_r^=$ of vertices with rank *exactly* r (for $r < \top$), one can simply compute the intersection $\bigcap_{1 \le k \le c/2} C^{2k-1}_{r_{2k-1}}$ of the corresponding sets C_x^i. However, in the algorithm we need the sets S_r containing all vertices v with a rank at least r and computing all sets $S_{r'}^=$ with $r' \ge r$ is not efficient. Towards a more efficient method to compute S_r, recall that a rank $r' < \top$ is higher than r if either (a) the right most odd element of r' is larger than the corresponding element in in r, i.e., $r'_{c-1} > r_{c-1}$, or (b) if r and r' coincide on the i right most odd elements, i.e., $r'_{c-2k+1} = r_{c-2k+1}$ for $1 \le k \le i$, and $r'_{c-2i-1} > r_{c-2i-1}$. In case (a) we can compute the corresponding vertices by $S_r^0 = \bigcup_{r_{c-1} < x \le n_{c-1}} C_x^{c-1}$ while in case (b) we can compute the corresponding vertices by $S_r^i = \bigcap_{1 \le k \le i} C^{c-2k+1}_{r_{c-2k+1}} \cap \bigcup_{r_{c-2i-1} < x \le n_{c-2i-1}} C_x^{c-2i-1}$ for $1 \le i \le c/2 - 1$. That is, we can reconstruct the set S_r by the following union of the above sets S_r^i, the set $S_r^=$ of vertices with rank r, and the set $S_\top$ of vertices with rank $\top$:

$$S_r = S_\top \cup S_r^= \cup \bigcup_{i=0}^{c/2-1} S_r^i$$

Hence, a set S_r can be computed with $O(c+n) \in O(n)$ many $\cup$ and $O(c)$ many $\cap$ operations; for the latter bound we use an additional set to store the set $\bigcap_{1 \le k \le i} C^{c-2k+1}_{r_{c-2k+1}}$ for the current value of i, such that for each set S_r^i we just need two $\cap$ operations. This implies the following lemma.

▶ **Lemma 4.** *Given the sets C_x^i as defined above, we can compute the set S_r that contains all vertices with rank at least r with $O(n)$ many symbolic set operations. No symbolic one-step operation is needed.*

Updating a set S_r. Now consider we have updated a set S_r during the iteration of the while-loop, and now we want to store the updated set S_r within the sets C_x^i. That is, we have already computed the fixed-point for S_r and are now in line 15 of the Algorithm. To this end, let S_r^{old} be the set as stored in C_x^i and S_r^{new} the updated set, which is a superset of the old one. First, one computes the difference S_r^{diff} between the two sets $S_r^{\text{diff}} = S_r^{\text{new}} \setminus S_r^{\text{old}}$; intuitively, the set S_r^{diff} contains the vertices for which the algorithm has increased the rank. Now for the vertices of S_r^{diff} we have to

(i) delete their old values by updating C_x^i to $C_x^i \setminus S_r^{\text{diff}}$ for each $i \in \{1, 3, \dots, c-1\}$ and each $x \in \{0, \dots, n_i\}$ and

(ii) store the new values by updating C_x^i to $C_x^i \cup S_r^{\text{diff}}$ for $i \in \{1, 3, \dots, c-1\}$ and $x = r_i$.

In total we have $O(c)$ many $\cup$ and $O(n)$ many $\setminus$ operations.

Notice that the update operation for a set S_r, as described above, also updates all sets $S_{r'}$ for $r' < r$. Thus, when using the more succinct representation via the sets C_x^i and executing SymbolicParityDominion literally, the computation of the maximal rank r' s.t. $S_{r'} \supseteq S_r$ would fail because of the earlier update of S_r. Hence, we have to postpone the update of S_r till the end of the iteration and adjust the computation of r' as follows. We do not update the set S'_r, and first compute the final value for r' by decrementing r' until $S_{r'} \supseteq S_r$ and then update S_r to S_r^{new} and thus implicitly also update the sets $S_{\tilde{r}}$ to $S_{\tilde{r}} \cup S_r$ for $r' < \tilde{r} < r$. This gives the following lemma.

▶ **Lemma 5.** *In each iteration of SymbolicParityDominion only $O(n)$ symbolic set operations are needed to update the sets C_x^i, and no symbolic one-step operation is needed.*

Number of Set Operations. To sum up, when introducing the succinct representation of the sets S_r, we only need additional $\cup$, $\cap$, and $\setminus$ operations, while the number of CPre_z

operations is unchanged. We show in Appendix A that whenever the algorithm computes or updates a set S_r, then we can charge a CPre_z operation for it, and each CPre_z operation is only charged for a constant number of set computations and updates. Hence, as both computing a set S_r and updating the sets C_x^i can be done with $O(n)$ set operations, the number of the additional set operations in SymbolicParityDominion is in $O(n \cdot \#\mathsf{CPre})$, for $\#\mathsf{CPre}$ being the number of CPre_z operations in the algorithm.

Putting Things Together. We presented a set-based symbolic implementation of the progress measure that uses $O(n)$ sets, $O(c \cdot n \cdot |M_h^\infty|)$ symbolic one-step operations and at most a factor of n more symbolic set operations. Using that $|M_h^\infty| \leq \binom{h+\lfloor c/2 \rfloor}{h} + 1$ we obtain the following key lemma that summarizes the result for computing dominions.

▶ **Key Lemma 6.** *For a parity game with n vertices and c priorities, and $h \in [1, n-1]$, SymbolicParityDominion computes a player-$\mathcal{E}$ dominion that contains all $\mathcal{E}$-dominions with at most $h+1$ vertices and can be implemented with $O\left(c \cdot n \cdot \binom{h+\lfloor c/2 \rfloor}{h}\right)$ symbolic one-step operations, $O\left(c \cdot n^2 \cdot \binom{h+\lfloor c/2 \rfloor}{h}\right)$ symbolic set operations, and $O(n)$ many sets.*

To solve parity games directly with Algorithm SymbolicParityDominion, we use the co-domain $M_{\mathcal{G}}^\infty$ instead of M_h^∞. Recall that we have $|M_{\mathcal{G}}^\infty| \in O\big(\big(\frac{n}{\lfloor c/2 \rfloor}\big)^{\lfloor c/2 \rfloor}\big)$ [29].

▶ **Theorem 7.** *Let $\xi(n,c) = \big(\frac{n}{\lfloor c/2 \rfloor}\big)^{\lfloor c/2 \rfloor}$. Algorithm SymbolicParityDominion computes the winning sets of parity games and can be implemented with $O\big(c \cdot n \cdot \xi(n,c)\big)$ symbolic one-step operations, $O\big(c \cdot n^2 \cdot \xi(n,c)\big)$ symbolic set operations, and $O(n)$ many sets.*

See the full version [12] for how to construct winning strategies within the same bounds.

4 Extensions and Conclusion

Big-Step Algorithm. We presented a set-based symbolic algorithm for computing a progress measure that solves parity games. Since the progress measure algorithm can also compute dominions of bounded size, it can be combined with the big step approach of [39] to improve the number of symbolic steps as stated in the following theorem.

▶ **Theorem 8.** *Let $\gamma(c) = c/3 + 1/2 - 4/(c^2-1)$ for odd c and $\gamma(c) = c/3 + 1/2 - 1/(3c) - 4/c^2$ for even c. There is a symbolic Big Step Algorithm that computes the winning sets for parity games and with the minimum of $O(n \cdot (\kappa \cdot n/c)^{\gamma(c)})$, for some constant κ, and $n^{O(\sqrt{n})}$ symbolic one-step operations and stores only $O(n)$ many sets.*

Concluding Remarks. In this work we presented improved set-based symbolic algorithms for parity games, and equivalently modal μ-calculus model checking. Our main contribution improves the symbolic algorithmic complexity of one of the most fundamental problems in the analysis of program logics, with numerous applications in program analysis and reactive synthesis. There are several practical approaches to solve parity games, such as, [15, 23, 27, 26, 3] and [42]. A practical direction of future work would be to explore whether our algorithmic ideas can be complemented with engineering efforts to obtain scalable symbolic algorithms for reactive synthesis of systems. An interesting theoretical direction of future work is to obtain set-based symbolic algorithms for parity games with quasi-polynomial complexity. The breakthrough result of [10] (see also [24]) relies on alternating poly-logarithmic space Turing machines. The follow-up papers of [30] and [21] that slightly improve the running

time and reduce the space complexity from quasi-polynomial to quasi-linear rely on succinct notions of progress measures. All these algorithms are non-symbolic, and symbolic versions of these algorithms are an open question, in particular encoding the novel succinct progress measures in the symbolic setting when storing at most a linear number of sets.

References

1 S. B. Akers. Binary decision diagrams. *IEEE Trans. Comput.*, C-27(6):509–516, 1978.

2 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *JACM*, 49:672–713, 2002. doi:10.1145/585265.585270.

3 M. Benerecetti, D. Dell'Erba, and F. Mogavero. Solving parity games via priority promotion. In *CAV*, pages 270–290, 2016. doi:10.1007/978-3-319-41540-6_15.

4 T. A. Beyene, S. Chaudhuri, C. Popeea, and A. Rybalchenko. A constraint-based approach to solving games on infinite graphs. In *POPL*, pages 221–234, 2014.

5 H. Björklund, S. Sandberg, and S. Vorobyov. A discrete subexponential algorithms for parity games. In *STACS*, pages 663–674, 2003. doi:10.1007/3-540-36494-3_58.

6 A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *TCS*, 178(1-2):237–255, 1997.

7 R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, C-35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.

8 J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138:295–311, 1969.

9 D. Bustan, O. Kupferman, and M. Y. Vardi. A measured collapse of the modal μ-calculus alternation hierarchy. In *STACS*, pages 522–533, 2004.

10 C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *STOC*, 2017. To appear.

11 P. Cerný, K. Chatterjee, T. A. Henzinger, A. Radhakrishna, and R. Singh. Quantitative synthesis for concurrent programs. In *CAV*, pages 243–259, 2011.

12 K. Chatterjee, W. Dvořák, M. Henzinger, and V. Loitzenbauer. Improved set-based symbolic algorithms for parity games. *CoRR*, abs/1706.04889, 2017. URL: https://arxiv.org/abs/1706.04889.

13 K. Chatterjee, M. Henzinger, and V. Loitzenbauer. Improved Algorithms for One-Pair and k-Pair Streett Objectives. In *LICS*, pages 269–280, 2015. doi:10.1109/LICS.2015.34.

14 A. Church. Logic, arithmetic, and automata. In *ICM*, pages 23–35, 1962.

15 L. de Alfaro and M. Faella. An accelerated algorithm for 3-color parity games with an application to timed games. In *CAV*, pages 108–120, 2007.

16 L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *CONCUR*, pages 142–156, 2003.

17 L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In *EMSOFT*, pages 148–165. 2001. doi:10.1007/3-540-45449-7_11.

18 L. de Alfaro, T. A. Henzinger, and R. Majumdar. Symbolic algorithms for infinite-state games. In *CONCUR*, pages 536–550, 2001. doi:10.1007/3-540-44685-0_36.

19 E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *FOCS*, pages 368–377, 1991. doi:10.1109/SFCS.1991.185392.

20 E. A. Emerson and Ch.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *LICS*, pages 267–278, 1986.

21 J. Fearnley, S. Jain, S. Schewe, F. Stephan, and D. Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. To be announced at SPIN, 2017. URL: http://arxiv.org/abs/1703.01296.

22 O. Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *LICS*, pages 145–156, 2009. doi:10.1109/LICS.2009.27.

23 O. Friedmann and M. Lange. Solving parity games in practice. In *ATVA*, pages 182–196, 2009.

24 H. Gimbert and R. Ibsen-Jensen. A short proof of correctness of the quasi-polynomial time algorithm for parity games, 2017. URL: https://arxiv.org/abs/1702.01953.

25 T. A. Henzinger, O. Kupferman, and S. K. Rajamani. Fair simulation. *Information and Computation*, 173(1):64–81, 2002. doi:10.1006/inco.2001.3085.

26 P. Hoffmann and M. Luttenberger. Solving parity games on the GPU. In *ATVA*, pages 455–459, 2013.

27 M. Huth, J. Huan-Pu Kuo, and N. Piterman. Concurrent small progress measures. In *HVC*, pages 130–144, 2011.

28 B. Jobstmann, A. Griesmayer, and R. Bloem. Program repair as a game. In *CAV*, pages 226–238, 2005. doi:10.1007/11513988_23.

29 M. Jurdziński. Small Progress Measures for Solving Parity Games. In *STACS*, pages 290–301, 2000. doi:10.1007/3-540-46541-3_24.

30 M. Jurdziński and R. Lazić. Succinct progress measures for solving parity games. To be announced at LICS, 2017. URL: https://arxiv.org/abs/1702.05051.

31 M. Jurdziński, M. Paterson, and U. Zwick. A Deterministic Subexponential Algorithm for Solving Parity Games. *SIAM J. Comput.*, 38(4):1519–1532, 2008.

32 C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Techn. J.*, 38(4):985–999, 1959. doi:10.1002/j.1538-7305.1959.tb01585.x.

33 P. Madhusudan and P. S. Thiagarajan. Distributed controller synthesis for local specifications. In *ICALP*, pages 396–407, 2001. doi:10.1007/3-540-48224-5_33.

34 R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993. doi:10.1016/0168-0072(93)90036-D.

35 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989. doi:10.1145/75277.75293.

36 P. J. Ramadge and W. Murray Wonham. Supervisory control of a class of discrete-event processes. *SIAM J. Control Optim.*, 25(1):206–230, 1987. doi:10.1137/0325013.

37 S. Safra. On the complexity of ω-automata. In *FOCS*, pages 319–327, 1988.

38 S. Safra. *Complexity of automata on infinite objects*. PhD thesis, Weizmann Inst., 1989.

39 S. Schewe. Solving Parity Games in Big Steps. *JCSS*, 84:243–262, 2017.

40 H. Seidl. Fast and simple nested fixpoints. *IPL*, 59(6):303–308, 1996.

41 F. Somenzi. CUDD: CU decision diagram package release 3.0.0, 2015. URL: http://vlsi.colorado.edu/~fabio/CUDD/.

42 S. Vester. Winning cores in parity games. In *LICS*, pages 662–671, 2016.

43 J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *CAV*, pages 202–215, 2000. doi:10.1007/10722167_18.

44 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135–183, 1998.

A Technical Appendix: Details for Section 3

Correctness. The correctness of Algorithm SymbolicParityDominion, stated in the following lemma, follows from combining Lemma 10 with Lemmata 11–13, which we prove below.

▶ **Lemma 9** (Correctness). *Algorithm SymbolicParityDominion computes the progress measure for a given parity game (with n vertices) and a given set of possible ranks M_h^∞ (for some integer $h \in [1, n-1]$).*

▶ **Lemma 10.** *Assuming that Invariant 3 holds, the ranking function $\rho_{\{S_r\}_r}$ induced by the family of sets $\{S_r\}_r$ at termination of Algorithm SymbolicParityDominion is equal to the progress measure for the given parity game and the co-domain M_h^∞.*

Proof. Recall that the progress measure is the least simultaneous fixed point of all $\mathrm{Lift}(., v)$-operators for the given parity game (where inc, dec, and the ordering of ranks are w.r.t. the given co-domain) and let the progress measure be denoted by $\tilde{\rho}$. Let $\{S_r\}_r$ be the sets in the algorithm at termination. For all $v \in V$ the ranking function $\rho_{\{S_r\}_r}(v)$ is defined as $\max\{r \in M_h^\infty \mid v \in S_r\}$. By Invariant 3(2) we have $\rho_{\{S_r\}_r}(v) \leq \tilde{\rho}(v)$ for all $v \in V$.

When the algorithm terminates, with $r = \top$, we have by Invariant 3(3) $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) = \rho_{\{S_r\}_r}(v)$ for each vertex v and thus $\rho_{\{S_r\}_r}$ is a simultaneous fixed point of the $\mathrm{Lift}(., v)$-operators. Now, as $\tilde{\rho}$ is the least simultaneous fixed point of all $\mathrm{Lift}(., v)$-operators, we obtain $\rho_{\{S_r\}_r}(v) \geq \tilde{\rho}(v)$ for all $v \in V$. Hence we have $\rho_{\{S_r\}_r}(v) = \tilde{\rho}(v)$ for all $v \in V$. ◀

▶ **Lemma 11.** *Before and after each iteration of the while-loop in Alg. SymbolicParityDominion we have $S_{r_1} \supseteq S_{r_2}$ for all $r_1 \leq r_2$ with $r_1, r_2 \in M_h^\infty$, i.e., Invariant 3(1) holds.*

Proof. The proof is by induction over the iterations of the while-loop. The claim is satisfied when we first enter the while-loop and only $S_{\bar{0}}$ is non-empty. It remains to show that when the claim is valid at the beginning of a iteration then the claim also hold afterwards. By the induction hypothesis, the sets $S_{r'}$ for $r' < r$ are monotonically decreasing. Thus it is sufficient to find the lowest rank r^* such that for all $r^* \leq r' < r$ we have $S_r \not\subseteq S_{r'}$ and add the vertices newly added to S_r to the sets $S_{r'}$ with $r^* \leq r' < r$, which is done in lines 16–25 of the while-loop. ◀

▶ **Lemma 12.** *Let $\tilde{\rho}$ be the progress measure of the given parity game and let $\rho_{\{S_r\}_r}(v) = \max\{r \in M_h^\infty \mid v \in S_r\}$ be the ranking function with respect to the family of sets $\{S_r\}_r$ that is maintained by the algorithm. Throughout Algorithm SymbolicParityDominion we have $\tilde{\rho}(v) \geq \rho_{\{S_r\}_r}(v)$ for all $v \in V$, i.e., Invariant 3(2) holds.*

Proof. We show the lemma by induction over the iterations of the while-loop. Before the first iteration of the while-loop only $S_{\bar{0}}$ is non-empty, thus the claim holds by $\tilde{\rho} \geq \bar{0}$.

Assume we have $\rho_{\{S_r\}_r}(v) \leq \tilde{\rho}(v)$ for all $v \in V$ before an iteration of the while-loop. We show that $\rho_{\{S_r\}_r}(v) \leq \tilde{\rho}(v)$ also holds during and after the iteration of the while-loop. As the update of $S_{r'}$ in line 22 does not change $\rho_{\{S_r\}_r}$, we only have to show that the invariant is maintained by the update of S_r in lines 4–14. Further $\rho_{\{S_r\}_r}(v)$ only changes for vertices newly added to S_r, thus we only have to take these vertices into account.

Let ℓ be the maximal index such that $r = \langle r \rangle_\ell$ or the highest odd priority if $r = \top$. Assume $r < \top$, the argument for $r = \top$ is analogous. The algorithm adds vertices to S_r in (1) line 6 and (2) line 8. In case (1) we add the vertices $\bigcup_{1 \leq k \leq (\ell+1)/2} (\mathsf{CPre}_{\mathcal{O}}(S_{\mathrm{dec}_{2k-1}(r)}) \cap P_{2k-1})$ to S_r. Let $v \in \mathsf{CPre}_{\mathcal{O}}\ (S_{\mathrm{dec}_{2k-1}(r)}) \cap P_{2k-1}$ for some $1 \leq k \leq (\ell+1)/2$.

- If $v \in V_{\mathcal{E}} \cap P_{2k-1}$, then all successors w of v are in $S_{\mathrm{dec}_{2k-1}(r)}$ and thus, by the induction hypothesis, have $\tilde{\rho}(w) \geq \mathrm{dec}_{2k-1}(r)$. Now as $v \in P_{2k-1}$, it has rank $\tilde{\rho}(v)$ at least $\mathrm{inc}_{2k-1}(\mathrm{dec}_{2k-1}(r)) = r$.
- If $v \in V_{\mathcal{O}} \cap P_{2k-1}$, at least one successors w of v is in $S_{\mathrm{dec}_{2k-1}(r)}$ and thus, by the induction hypothesis, has $\tilde{\rho}(w) \geq \mathrm{dec}_{2k-1}(r)$. Now as $v \in P_{2k-1}$, it has rank $\tilde{\rho}(v)$ at least $\mathrm{inc}_{2k-1}(\mathrm{dec}_{2k-1}(r)) = r$.

For case (2) consider a vertex $v \in \mathsf{CPre}_{\mathcal{O}}(S_r) \setminus \bigcup_{\ell < k \leq d} P_k$ added in line 8.

- If $v \in V_{\mathcal{E}}$, all successors w of v are in S_r and thus, by the induction hypothesis, have $\tilde{\rho}(w) \geq r$. Since the priority of v is $\leq \ell$, we have $\tilde{\rho}(v) \geq \langle r \rangle_\ell = r$.

- If $v \in V_{\mathcal{O}}$, at least one successors w of v is in S_r and thus, by the induction hypothesis, has $\tilde{\rho}(w) \geq r$. Since the priority of v is $\leq \ell$, we have $\tilde{\rho}(v) \geq \langle r \rangle_\ell = r$. ◀

▶ **Lemma 13.** *Before and after each iteration of the while loop we have for the rank stored in r and all vertices v either* $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) \geq r$ *or* $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) = \rho_{\{S_r\}_r}(v)$. *At line 15 of the algorithm we additionally have $v \in S_r$ for all vertices v for which the value of* $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v)$ *is equal to r. Thus Invariant 3(3) holds.*

Proof. We show the claim by induction over the iterations of the while-loop. Before we first enter the loop, we have $r = \mathrm{inc}(\bar{0})$ and $S_{\bar{0}} = V$ and thus the claim is satisfied. For the inductive step, let r^{old} be the value of r and ρ^{old} the ranking function $\rho_{\{S_r\}_r}$ before a fixed iteration of the while-loop and assume we have for all $v \in V$ either $\mathrm{Lift}(\rho^{\mathrm{old}}, v)(v) \geq r^{\mathrm{old}}$ or $\mathrm{Lift}(\rho^{\mathrm{old}}, v)(v) = \rho^{\mathrm{old}}(v)$ before the iteration of the while-loop. Let r^{new} be the value of r and ρ^{new} the ranking function $\rho_{\{S_r\}_r}$ after the iteration. We have three cases for the value of r^{new}:

1. $r^{\mathrm{new}} = \mathrm{inc}(r^{\mathrm{old}})$ (line 16),
2. $r^{\mathrm{new}} = r^{\mathrm{old}} = \top$ (line 18), or
3. $r^{\mathrm{new}} < r^{\mathrm{old}}$, i.e., the rank is decreased in lines 21–25 to maintain anti-monotonicity.

We show in Claim 14 that, in all three cases, if a set $S_{r'}$, for some $r' < r^{\mathrm{old}}$, is not changed in the considered iteration of the while-loop then for all $v \in V$ with $\mathrm{Lift}(\rho^{\mathrm{new}}, v)(v) \leq r'$ we have that $\mathrm{Lift}(\rho^{\mathrm{new}}, v)(v) = \rho^{\mathrm{new}}(v)$.

Given Claim 14, we prove the first part of the invariant as follows. In the case (1) the lowest (and only) rank for which the set is updated is r^{old}, thus it remains to show $\mathrm{Lift}(\rho^{\mathrm{new}}, v)(v) = \rho^{\mathrm{new}}(v)$ for vertices with $\mathrm{Lift}(\rho^{\mathrm{new}}, v)(v) = r^{\mathrm{old}}$, which is done by showing the second part of the invariant, namely that $v \in S_{r^{\mathrm{old}}}$ for all vertices v with $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) = r^{\mathrm{old}}$ after the update of the set $S_{r^{\mathrm{old}}}$ in lines 4–14; for case (1) we have $\rho^{\mathrm{new}} = \rho_{\{S_r\}_r}$ at this point.

In the cases (2) and (3) we have that the lowest rank for which the set is updated in the iteration is equal to r^{new}, thus Claim 14 implies that the invariant $\mathrm{Lift}(\rho^{\mathrm{new}}, v)(v) \geq r^{\mathrm{new}}$ or $\mathrm{Lift}(\rho^{\mathrm{new}}, v)(v) = \rho^{\mathrm{new}}(v)$ holds for all $v \in V$ after the while-loop.

▶ **Claim 14.** *Let $r^* \leq r^{old}$ be a rank with the guarantee that no set corresponding to a lower rank than r^* is changed in this iteration of the while-loop. Then we have for all $v \in V$ either* $\mathrm{Lift}(\rho^{new}, v)(v) \geq r^*$ *or* $\mathrm{Lift}(\rho^{new}, v)(v) = \rho^{new}(v)$ *after the iteration of the while-loop.*

To prove the claim, note that since each set S_r is monotonically non-decreasing over the algorithm, we have $\rho^{\mathrm{new}}(v) \geq \rho^{\mathrm{old}}(v)$ and $\mathrm{Lift}(\rho^{\mathrm{new}}, v)(v) \geq \rho^{\mathrm{new}}(v)$. Assume by contradiction that there is a vertex v with $\mathrm{Lift}(\rho^{\mathrm{new}}, v)(v) > \rho^{\mathrm{new}}(v)$ and $\mathrm{Lift}(\rho^{\mathrm{new}}, v)(v) < r^*$. The latter implies $\mathrm{best}(\rho^{\mathrm{new}}, v) < r^*$. By the induction hypothesis for $r^* \leq r^{\mathrm{old}}$ we have $\mathrm{Lift}(\rho^{\mathrm{old}}, v)(v) = \rho^{\mathrm{old}}(v)$. By $\rho^{\mathrm{new}}(v) \geq \rho^{\mathrm{old}}(v)$ and the definition of the lift-operator this implies $\mathrm{best}(\rho^{\mathrm{new}}, v) > \mathrm{best}(\rho^{\mathrm{old}}, v)$, i.e., the rank assigned to at least one vertex w with $(v, w) \in E$ is increased. By $\mathrm{best}(\rho^{\mathrm{new}}, v) < r^*$ this implies that a set $S_{r'}$ with $r' < r^*$ is changed in this iteration, a contradiction to the definition of r^*. Hence, the claim holds.

It remains to show that $v \in S_{r^{\mathrm{old}}}$ for all vertices v with $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) = r^{\mathrm{old}}$ after the update of the set $S_{r^{\mathrm{old}}}$ in lines 4–14. Towards a contradiction assume that there is a $v \notin S_{r^{\mathrm{old}}}$ such that $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) = r^{\mathrm{old}}$. Assume $v \in V_{\mathcal{E}}$, the argument for $v \in V_{\mathcal{O}}$ is analogous. Let ℓ be maximal such that $r^{\mathrm{old}} = \langle r^{\mathrm{old}} \rangle_\ell$ for $r^{\mathrm{old}} < \top$ and let ℓ be the highest odd priority in the parity game for $r^{\mathrm{old}} = \top$. Notice that $\alpha(v)$ can be at most ℓ for $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) = r^{\mathrm{old}}$ to hold. We now distinguish two cases depending on whether $\alpha(v)$ is odd or even.

- If $\alpha(v)$ is odd, i.e., $\alpha(v) = 2k-1$ for some $k \leq (\ell+1)/2$, then we have that all successors w of v have $\rho_{\{S_r\}_r}(w) \geq \mathrm{dec}_{2k-1}(r^{\mathrm{old}})$ and thus, by Lemma 11, $w \in S_{\mathrm{dec}_{2k-1}(r^{\mathrm{old}})}$. But then v would have being included in $S_{r^{\mathrm{old}}}$ in line 6, a contradiction.
- If $\alpha(v)$ is even, i.e., $\alpha(v) = 2k$ for some $k \leq \ell/2$, then we have that all successors w of v have $\rho_{\{S_r\}_r}(w) \geq r^{\mathrm{old}}$. Then by the definition of $\rho_{\{S_r\}_r}$ it must be that $w \in S_{r'}$ for some $r' \geq r^{\mathrm{old}}$ and by Lemma 11 it must be that $w \in S_{r^{\mathrm{old}}}$. But then v would have being included in $S_{r^{\mathrm{old}}}$ in line 8, a contradiction.

Thus, after the update of the set $S_{r^{\mathrm{old}}}$ we have that $v \in S_{r^{\mathrm{old}}}$ for all vertices v with $\mathrm{Lift}(\rho_{\{S_r\}_r}, v)(v) = r^{\mathrm{old}}$. Together we the above observations this proves the lemma. ◀

Number of symbolic operations. We address next the number of symbolic operations of Algorithm SymbolicParityDominion when using the sets S_r directly. We analyze the number of symbolic operations when using a linear number of sets below. The main idea is that a set S_r is only reconsidered if at least one new vertex was added to S_r.

▶ **Lemma 15.** *For parity games with n vertices and c priorities Algorithm SymbolicParityDominion takes $O(c \cdot n \cdot |M_h^\infty|)$ many symbolic operations and uses $O(|M_h^\infty|)$ many sets, where h is some integer in $[1, n-1]$.*

Proof. In the algorithm we use one set S_r for each $r \in M_h^\infty$ and thus $|M_h^\infty|$ many sets. We first consider the number of symbolic operations needed to compute the sets S_r in lines 4–14, and then the number of symbolic operations to compute the new value of r in lines 15–25.

1. Whenever we consider a set S_r, we first initialize the set with $O(c)$ many symbolic operations (lines 6 & 11). After that we do a fixed-point computation that needs symbolic operations proportional to the number of added vertices. Now fix a set S_r and consider all the fixed-point computations for S_r over the whole algorithm. As only $O(n)$ many vertices can be added to S_r, all these fixed-points can be computed in $O(n+\#r)$ symbolic operations, where $\#r$ is the number of times the set S_r is considered (the algorithm needs a constant number of symbolic operations to realize that a fixed-point was already reached). Each set S_r is considered at least once and only reconsidered when some new vertices are added to the set, i.e., it is considered at most n times. Thus for each set S_r we have $O(c \cdot n)$ many operations, which gives a total number of operations of $O(c \cdot n \cdot |M_h^\infty|)$.
2. Now consider the computation of the new value of r in lines 15–25. Lines 15–19 take a constant number of operations. It remains to count the iterations of the repeat-until loop in lines 20–25, which we bound by the number of iterations of the while-loop as follows. Whenever a set $S_{r'}$ is considered as the left side argument in line 24, then the new value for r is less or equal to $\mathrm{inc}(r')$ and thus there will be another iteration of the while-loop considering $\mathrm{inc}(r')$. As there are only $O(n \cdot |M_h^\infty|)$ many iterations of the while-loop over the whole algorithm, there are only $O(n \cdot |M_h^\infty|)$ many iterations of the repeat-until loop in total. In each iteration a constant number of operations is performed.

By 1. and 2. we have that Algorithm SymbolicParityDominion takes $O(c \cdot n \cdot |M_h^\infty|)$ many symbolic operations. ◀

Number of set operations in linear space algorithm. For the proof of Lemma 6 and thus of Theorem 7 it remains to show that whenever the algorithm computes or updates a set S_r using the succinct representation with the sets C_x^i introduced in Section 3.2, then we can charge a CPre_z operation for it, and each CPre_z operation is only charged for a constant number of set computations and updates. The argument is as follows.

(a) Whenever the algorithm computes a set S_r in line 6 or 11, at least one CPre_z computation with this set is done.

(b) Now consider the computation of the new value of r. The subset tests in lines 16 and 18 are between a set that was already computed in line 6 or 11 and the set computed in line 8 or 13 and thus, if we store these sets, we do not require additional operations. Whenever a set $S_{r'}$ is considered as the left side argument in line 24, then the new value of r is less or equal to $\mathrm{inc}(r')$ and thus there will be another iteration of the while-loop considering $\mathrm{inc}(r')$. Hence, we can charge the additional operations needed for the comparison to the CPre_z operations of the next iteration that processes the rank $\mathrm{inc}(r')$.

(c) Finally, we only need to update the sets C^i_x once per iteration and in each iteration we perform at least one CPre_z computation that we can charge for the update.